форма № Н-9.02

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-методичний комплекс
«Інститут післядипломної освіти» КПІ імені Ігоря Сікорського

Кафедра системного програмування і спеціалізованих комп'ютерних систем

«До захисту допущено»
Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ
(підпис)
"___"_____20__ р.

Дипломний проект
на здобуття ступеня бакалавра

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: Алгоритм та програма розрахунку складності GL-моделі, що перетворюється

Виконав: слухач групи ЗКІ-зп71

БОЛДИР Євген Геннадійович _____
(підпис)

Керівник: д.т.н., доц. Віталій РОМАНКЕВИЧ _____
(підпис)

Консультант з нормоконтролю: к.т.н., доц. Ярослав КЛЯТЧЕНКО _____
(підпис)

Рецензент: _____
(підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Слухач _____
(підпис)

Київ – 2020

форма № Н-9.01

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-методичний комплекс «Інститут післядипломної освіти» КПІ ім. Ігоря Сікорського

(повна назва)

Кафедра системного програмування і спеціалізованих комп'ютерних систем

(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність **123 «Комп'ютерна інженерія»**

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис)
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проект слухача

БОЛДИР Євген Геннадійович

1. Тема проекту «Алгоритм та програма розрахунку складності GL-моделі, що перетворюється», керівник проекту д.т.н., доц. Віталій РОМАНКЕВИЧ, затверджені наказом по університету від «___»_____ 20__ р. №_____
2. Строк подання студентом проекту _____
3. Вихідні дані до проекту див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень та обґрунтування теми дипломного проекту;
   - оцінка складності моделі;
   - опис програмного продукту.

5. Перелік графічного матеріалу:
   - схема алгоритму побудови дерева реберних функцій;
   - схема алгоритму розподілу кольорів;
   - схема алгоритму евристичного розподілу кольорів;
   - структурна схема програми розрахунку складності.

6. Консультанти:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	к.т.н. доц. Ярослав КЛЯТЧЕНКО		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Строк виконання етапів роботи	Примітка
1	Вивчення літератури за тематикою роботи		
2	Складання та узгодження технічного завдання		
3	Огляд існуючих рішень		
4	Аналіз існуючих рішень. Розробка алгоритмів		
5	Програмне тестування алгоритмів		
6	Підготовка графічної частини дипломного проекту		
7	Оформлення документації дипломного проекту		
8	Попередній захист матеріалів диплому на кафедрі		

Слухач _____ Євген БОЛДИР

(підпис)

Керівник проекту _____ Віталій РОМАНКЕВИЧ

(підпис)

Поз.	Формат	Позначення	Найменування	К-ть. арк.		.
			Документація загальна			
			Знову розроблена			
	А4	ІАЛЦ.466500.002 ТЗ	Алгоритм та програма розрахунку	5		
			складності GL-моделі, що перетворюється.			
			Технічне завдання			
	А4	ІАЛЦ.466500.003 ТП	Алгоритм та програма розрахунку	1		
			складності GL-моделі, що перетворюється.			
			Відомість технічного проекту			
	А4	ІАЛЦ.466500.004 ПЗ	Алгоритм та програма розрахунку	53		
			складності GL-моделі, що перетворюється.			
			Пояснювальна записка			
	А1	ІАЛЦ.466500.005 Д1	Дерево реберних функцій.	1		
			Схема алгоритму			
	А1	ІАЛЦ.466500.006 Д2	Розподіл кольорів.	1		
			Схема алгоритму			
	А1	ІАЛЦ.466500.007 Д3	Евристичний розподіл кольорів	1		
			Схема алгоритму			
	А1	ІАЛЦ.466500.008 Д4	Програма розрахунку складності	1		
			Схема структурна			
		Диск CD	Матеріали роботи	1		

					ІАЛЦ.466500.001 ОА		
Зм..	Арк	№ докум.	Підпис	Дата			
Розроб.		Болдир Є.Г.			Алгоритм та програма розрахунку складності GL-моделі, що перетворюється. Опис альбому.	Літ.	Аркуш 1 / Аркушів 1
Перев.		Романкевич В.О.					
Реценз.						НМК «ІПО», гр. ЗКІ-зп71	
Н. контр.		Клятченко Я.М.					
Затверд.		Романкевич В.О.					

ЗМІСТ

стор.

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування розробки «Алгоритм та програма розрахунку складності GL-моделі, що перетворюється».

Програмний продукт використовується для оцінки і розрахунку складності графологічної моделі (GL-моделі), яка в свою чергу використовується для розрахунку надійності відмовостійких багатомодульних систем (ВБС).

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є індивідуальне завдання на здобуття ступеня бакалавра зі спеціальності 123 «Комп’ютерна інженерія», затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення алгоритмів та програмного продукту, що дозволяє провести оцінку та розрахунок складності GL-моделі на початкових стадіях її розробки та аналіз отриманих результатів з метою виявлення можливих закономірностей.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є науково-технічна література і публікації в періодичних виданнях з питанню розрахунку надійності відмовостійких

багатопроцесорних систем (ВБС), побудови GL-моделей для розрахунку надійності ВБС, базові та небазові GL-моделі, а також література присвячена проблемі розфарбування графів.

Також джерелами є документація та література по середовищу розробки програмного продукту: мови програмування C#, платформи .NET.

5. ТЕХНІЧНІ ВИМОГИ

- Персональним завданням даної бакалаврської кваліфікаційної роботи є проектування та розробка програмного продукту, який надає можливість зважити складність перетворення GL-моделі, яка використовується для розрахунку надійності відмовостійких багатопроцесорних систем, на початкових етапах її проектування.
- Програма має виконати розрахунок та забезпечити результат за не дуже великий проміжок часу.
- Повинен бути спроектований та розроблений алгоритм повного пошуку, який дає конкретні значення оцінки складності перетворення GL-моделі.
- Повинена бути спроектована, розроблена та виконана модель евристичного алгоритму оцінки складності перетворення GL-моделі, який буде розраховувати результат швидше відносно точного алгоритму.
- Повинні бути виконані засоби візуалізації знайденого варінту розфарбування графу.
- Повинна бути виконана можливість збереження інформації у файл.

6. ВИМОГИ ДО ТЕХНІЧНИХ ЗАСОБІВ

6.1 Вимоги до апаратного забезпечення

- Процесор: Intel Core I3, або вищий, або аналог інших виробників.
- Обсяг оперативної пам'яті: не нижче 1Gb.
- Вільний дисковий простір: не нижче 150Mb.

6.2 Вимоги до програмного забезпечення

- Операційна система сімейства Microsoft Windows не нижче версії XP, але краще 7, 10 або вищі.
- Мінімально необхідний дистрибутивний пакет .NET Framework 2.0 або вище.

7. ЕТАПИ РОЗРОБКИ

2. Складання та узгодження технічного завдання
3. Вивчення літератури
4. Розробка проекту
5. Відлагодження проекту
6. Оформлення документації дипломного проекту

Поз.	Формат	Позначення	Найменування	К-ть. арк.	№ екз.	
			Документація загальна			
			Знову розроблена			
	А4	ІАЛЦ.466500.004 ПЗ	Алгоритм та програма розрахунку	53		
			складності GL-моделі, що перетворюється.			
			Пояснювальна записка			
	А1	ІАЛЦ.466500.005 Д1	Дерево реберних функцій.	1		
			Схема алгоритму			
	А1	ІАЛЦ.466500.006 Д2	Розподіл кольорів.	1		
			Схема алгоритму			
	А1	ІАЛЦ.466500.007 Д3	Евристичний розподіл кольорів	1		
			Схема алгоритму			
	А1	ІАЛЦ.466500.008 Д4	Програма розрахунку складності	1		
			Схема структурна			
		Диск CD	Матеріали роботи	1		

Зм.	Арк	№ докум.	Підпис	Дата	ІАЛП.466500.003 ТП		
Розроб.		Болдир Є.Г.			Алгоритм та програма розрахунку складності GL-моделі, що перетворюється. Відомість технічного проекту	Лит.	Аркуш 1 / Аркушів 1
Перев.		Романкевич В.О.					
Реценз.						НМК «ІПО», гр. ЗКІ-зп71	
Н. контр.		Клятченко Я.М.					
Затверд.		Романкевич В.О.					

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ВБС – відмовостійкі багатомодульні (багатопроцесорні) системи

GL-модель – графо-логічна модель

ВСТУП

Розповсюдження відмовостійких багатопроцесорних систем (ВБС) набуває з кожним роком все більшого поширення, що пов’язано з можливостями одночасного підвищення як швидкодії, так і надійності. Сказане призводить до розширення області їх використання, що відноситься не тільки до високопродуктивних обчислювальних систем, але і до систем управління складними об’єктами, які потребують забезпечення високого рівня надійності таких систем.

Розрахунок надійності ВБС, які мають досить велику кількість процесорів, складну топологію зв’язків дуже часто призводить до значних алгоритмічних та обчислювальних труднощів.

Сказане стосується, зокрема, ВБС, які неоднаково реагують на появу відмов однакової кратності. Існує декілька методів розрахунку надійності подібних систем, серед яких статистичний метод, що базується на виконанні експериментів з моделями поведінки систем у потоці відмов, відрізняється своєю універсальністю. Реакцію ВБС на відмови своїх процесорів можуть відображати деякі математичні моделі, наприклад, GL-модель, особливістю якої є відносна простота формування самої моделі.

Одна з задач, що вирішуються на ранніх етапах проектування ВБС та GL-моделі, є оцінка її складності. Від складності моделі залежить час виконання лише одного експерименту з нею. Це означає, що за один і той же час з більш простою моделлю можна виконати більшу кількість експериментів і відповідно мати більшу точність розрахунку надійності ВБС, адже мова йде про статистичні методи.

Однією з складових критерію складності GL-моделі є кількість її ребер. Отже метою даної роботи була розробка алгоритму та програмного продукту,

який забезпечує розрахунок реберної складності моделі, обчислює границі, в межах яких буде знаходитись кількість ребер GL-моделі, що

слугує для розрахунку надійності ВБС, у першу чергу ймовірності її безвідмовної роботи за заданий час.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

1.1 Що таке GL-модель

Відмовостійкі багатопроцесорні обчислювальні та управляючі системи, як це вже відмічалося у вступі, з кожним роком набувають все більшого поширення, у тому числі в системах управління складними і відповідальними об'єктами (літаки, ракети, космічні об'єкти, складні технологічні процеси, фінансові розрахунки та інші), чия відмова може призвести до тяжких наслідків економічного та екологічного характеру або навіть до людських жертв. Це зобов'язує розробників таких систем використовувати різні апаратні та програмні методи і засоби підвищення надійності, зокрема, забезпечення їх відмовостійкості. Останнє досягається за рахунок реконфігурації, тобто забезпечення можливості виконання функцій управління, які виконував процесор, що вийшов з ладу, тим процесорам, що залишились роботоздатними. При цьому для обчислювальних систем характерні велика кількість процесорів з одного боку і регулярність структури їх з'єднань – з іншого. Це певною мірою полегшує їх проектування і розрахунок необхідних параметрів. Для систем управління навпаки - порівняно менша кількість процесорів (десятки, сотні, іноді тисячі) і відносна нерегулярність структури.

ВБС, як правило, можуть самотестуватися і реконфігуруватися. Проблема організації самотестування багатопроцесорних систем вирішується іноді дуже складно, але її забезпечення є необхідною складовою функціонування багатопроцесорних систем. Ця проблема тісно пов’язано з проблемою забезпечення високої надійності ВБС, але вирішується окремо у рамках теорії діагностування.

Нижче у якості моделі для розрахунку надійності відмовостійких багатопроцесорних систем будемо розглядати GL-модель, головною перевагою якої є поєднання властивостей графів і булевих функцій.

ВБС, що є стійкою до відмов певної кратності, будемо називати базовою. Це означає, що кількість нероботоздатних процесорів такої системи не повинно перевищувати деякої фіксованої величини, інакше система буде не в змозі продовжувати функціонування з тими ж параметрами за рахунок реконфігурації.

Систему, що має n модулів і стає нероботоздатною при відмові не менше будь-яких $k+1$ процесорів, за умови $k<n$, називатимемо базовою k –ВБС і позначати $K(k,n)$.

Графова модель для розрахунку надійності систем без резервування і інших засобів підвищення надійності (тобто невідмовостійка) добре відома: можна поставити у відповідність такій системі двополюсний граф, всі вершини якого послідовно зв’язані ребрами, причому кожне ребро відповідає одному модулю системи. Фактично це елементарний ланцюг (рис. 1), що дозволяє порівняно нескладно записати логічну структурну функцію системи і з її допомогою розрахувати її надійність.

Можна сказати, що в цьому випадку кожному ребру приписана індикаторна булева змінна x_i, де i=1..., n, де n – кількість процесорів системи, що визначає наявність або відсутність ребра і залежить від того, відмовив (x_i =0) або роботоздатний (x_i =1) відповідний модуль системи (рис. 1). Якщо хоч один процесор вийде з ладу, то в графі зникне ребро, і зв'язність між полюсами пропаде, що відповідає відмові всієї системи. Критерієм роботоздатності системи в такій моделі є існування шляху між двома виділеними полюсами графа. Ясно, що така система є 0-відмовостійкою системою (0-ВБС).

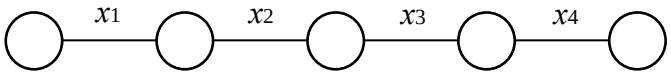


Рис. 1

Іноді (наприклад, для мережевих структур) в якості моделі для розрахунку надійності може бути вибраний багатополюсний граф. У такому випадку критерієм надійності буде зв'язність графа (система вважається роботоздатною тільки тоді, коли кожен вузол графа має зв'язок з будь-яким іншим вузлом). Фактично розрахунок надійності може бути зведений до визначення ймовірності події, що полягає в тому, що, принаймні, одне дерево графу зберігається. Очевидно, що при цьому комбінаторне завдання перебору всіх дерев для досить складних мереж призводить, як і у випадку з тими ж самими двополюсними графами, до значних обчислювальних труднощів.

Наведені приклади підтверджують актуальність та важливість задачі побудови нових більш простих моделей поведінки ВБС, які дозволили б зробити аналіз станів системи більш доступним і в результаті спростити розрахунки і отримати достатньо точну оцінку параметрів надійності багатопроцесорних систем.

Відзначимо, що для ефективного аналізу поведінки систем у потоці відмов, що мають велику кількість станів, доволі важливим є компактне представлення критерію, відповідно до якого здійснюється оцінка реакції системи в певних умовах.

Для відображення стану системи GL-модель використовує відповідність між зв'язністю деякого графа і обчислюваними для кожного стану системи значеннями певним чином вибраних булевих функцій. Проте на відміну від відомих моделей (наприклад, двополюсних), що

використовують графи, в даному випадку граф, що використовується при побудові моделі, може мати набагато простішу структуру, ніж структура модельованої системи. Більше того, топологія зв'язків процесорів між собою в системі аж ніяк не впливає на структуру графа. Це досягається за рахунок «перенесення» складності формування моделі на булеві функції, що приписуються ребрам графу.

Опишемо коротко основні принципи побудови графологічній моделі. Відзначимо, що головна ідея полягає в забезпеченні можливості змінювати структуру графа відповідно до вектора стану елементів системи, що досягається шляхом приписування ребрам графа певних булевих функцій.

Поставимо у відповідність *k*-ВБС деякий зв'язний неорієнтований граф G, ребра якого позначаються вибраними певним чином булевими функціями $f_1(x_1,\ldots,x_n),\ldots,f_L(x_1,\ldots,x_n)$, де кожна змінна x_i $(i=\overline{1,n})$ є індикаторною функцією, яка позначає стан *i*-го елементу системи, тобто $x_i=1$, якщо *i*-й елемент системи працездатний, і $x_i=0$ якщо *i*-й елемент системи відмовив. Ребро, позначене функцією $f_i(x_1,\ldots,x_n)$, залишається у графі G, якщо $f_i(x_1,\ldots,x_n)=1$ і видаляється з нього, якщо $f_i(x_1,\ldots,x_n)=0$.

Якщо вибрати функції $f_1(x_1,\ldots,x_n),\ldots,f_L(x_1,\ldots,x_n)$ таким чином, щоб граф G втрачав зв'язність тільки при рівності нулю не менше $k+1$ будь-яких з n змінних, то відмову *n*-модульної системи можна інтерпретувати, як втрату зв'язності графа G. Іншими словами, функції $f_1,\ldots,f_L$ повинні мати наступну властивість: для всіх *n*-розрядних наборів, що містять будь-яке поєднання з не менше, ніж $k+1$ нульових розрядів, знайдеться r $(r < L)$ функцій з множини $f_1,\ldots,f_L$, які приймають нульові значення на цих наборах і позначають r таких ребер, видалення яких з графа G призводить до втрати зв'язності.

Очевидно, що досить важливими етапами при побудові графологічної моделі є вибір графу G і функцій $f_1, \ldots, f_L$.

Найпростішим графом є незамкнутий ланцюг, а найпростішими функціями - $f_i = x_i$. Такий граф з ребрами, відміченими функціями $f_i = x_i$, може бути використаний в графологічній моделі для 0 – ВБС. Якщо ланцюг замкнути, тобто отримати циклічний граф, то разом з цими ж функціями $f_i = x_i$ отриманий граф стане графологічною моделлю 1 – ВБС (рис. 2).

Таким чином вище були описані графологічні моделі 0 – ВБС та 1 – ВБС. Проте починаючи з 2 – ВБС доволі складно узагальнити запропонований вище алгоритм побудови GL-моделі.

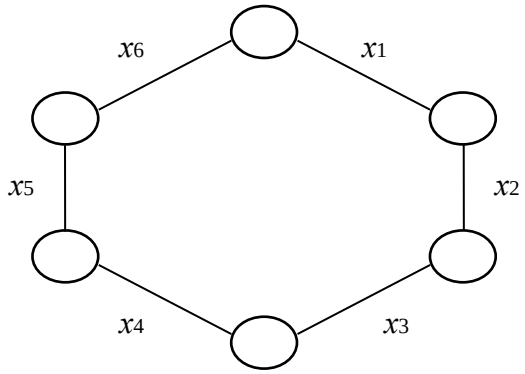


Рис. 2

Існують декілька інших способів побудови графологічної моделі для довільної *k*-ВБС. Нагадаємо, що *k*-ВБС це така відмовостійка багатомодульна система, яка стає непрацездатною при відмові не менше ніж *k+1* модулів. Усі ці методи будуються на основі циклічного неорієнтованого графу, який був запропонований як графологічна модель для 1 – ВБС (рис. 2). Якщо спроектувати основний принцип для такого циклічного графу, який має реалізовувати графологічну модель для *k*-ВБС, то він буде наступним:

при появі не менше ніж *k* відмов, у графі повинно випадати не менше ніж 2 ребра, тобто не менше ніж дві реберні функції $f_i(x_1,\ldots,x_n)$ повинні бути рівними 0; при появі меншій або рівній *k* кількості відмов з графу повинно випадати не більше ніж одне ребро. Дійсно, при випаданні лише одного ребра циклічний граф залишається зв'язним, що є критерієм роботоздатності моделі, проте виключення з графу вже двох ребер призводить до втрачання ним зв'язності.

Отже структура графу є зрозумілою. Наступною проблемою при побудові GL-моделі на основі циклічного графу є вибір такої кількості ребер і таких реберних функції, які б задовольняли описаному вище принципу. Як вже було сказано, для моделювання 1 – ВБС необхідно взяти n ребер кожному з яких буде приписна функція $f_i = x_i$.

Розглянемо нижче алгоритм побудови мінімізованої GL-моделі, а конкретніше алгоритм побудови реберних функцій для базової *k*-ВБС.

Ідея цього методу полягає в послідовному розділенні множини компонентів системи на дві рівних або майже рівних непересічних підмножини і розподілі по ним поточної кількості відмов, до яких система повинна бути стійка. Цей процес продовжується до тих пір, доки кількість відмов або не буде рівна потужності підмножини, або не буде рівною одиниці. Далі записується функція, відповідна кожному такому розподілу, і приписується одному ребру графа моделі. В результаті виходить, так звана, канонічна GL-модель. Після чого, отриману канонічну GL-модель можна мінімізувати, за рахунок операцій склеювання реберних функції. В даній роботі не є важливим процес побудови реберних функцій, тому алгоритм мінімізації описаний не буде. Важливим буде сказати, що отримана після процесу мінімізації модель втрачає мінімальну кількість ребер при появі вектора стану системи з *m+1* нулем, тобто рівно 2 ребра.

Набагато наочніше зрозуміти процес побудови мінімізованої базової GL-моделі для *K*(*m*,*n*) ВБС можна розглянувши так зване дерево ієрархії реберних функцій, в якому кожній вершині дерева відповідає певне ребро GL-моделі.

Побудоване дерево ієрархії реберних функції (надалі дерево) для *K(4,13)* зображене на рис. 3. Кожна вершина дерева (не вузол) відповідає ребру GL-моделі для відмовостійкої багатомодульної системи *K*(*4,13*). З рисунку видно, що це наступні 10 ребер:

- *K(3,7) ?K(1,6)*
- *K(2,7) ?K(2,6)*
- *K(1,7) ?K(3,6)*
- *K(4,4)*
- *K(3,4) ?K(1,3)*
- *K(2,4) ?K(2,3)*
- *K(1,4) ?K(3,3)*
- *K(3,3) ?K(1,3)*
- *K(2,3) ?K(2,3)*
- *K(1,3) ?K(3,3)*

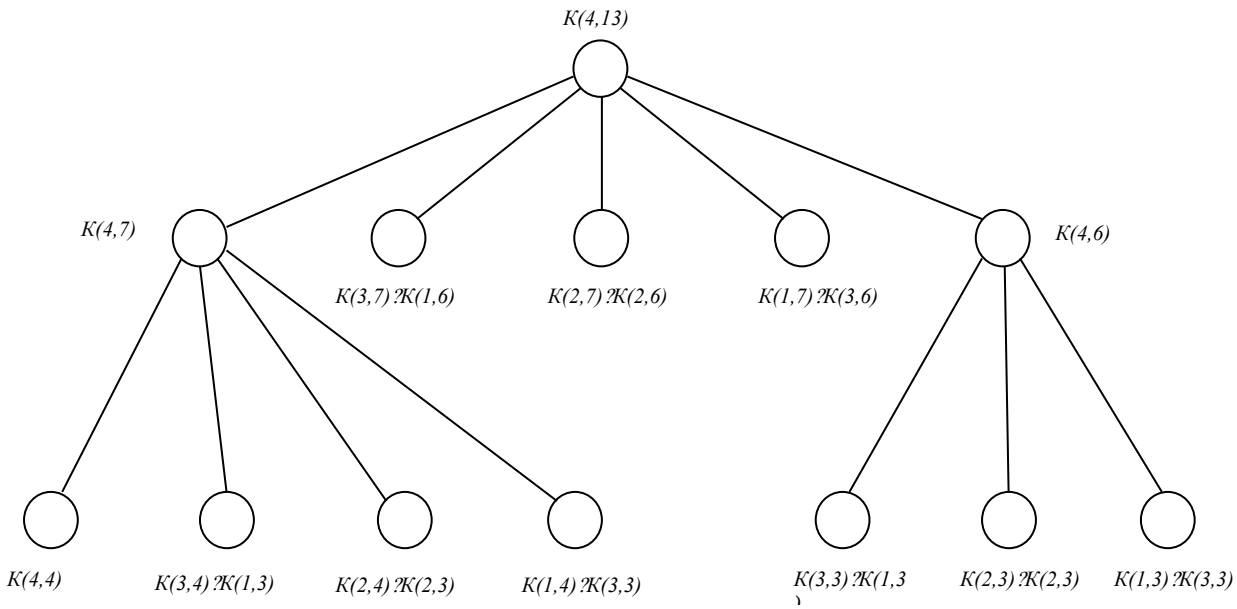


Рис. 3. Дерево ієрархії для *K*(*4,13*)

Сама побудова дерева виконується за наступним рекурсивним алгоритмом.

Позначимо через *K'(m,n)* множину реберних функцій GL-моделі *K(m,n)*. Кожен елемент з *K'(m,n)* присвоєний лише одному ребру моделі. Процес отримання елементів *K'(m,n)* можна представити у вигляді рекурсивного співвідношення:

$$K'(m_1,n_1)=>K'(m_1\text{-}i,[n_1/2])\ ?K'(i,]n_1/2[) \qquad (1)$$

де $i=0..m_1$, $n_1>m_1$, $m_1\text{-}i\leq[n_1/2]$, $i\leq]n_1/2[$, $[x]$ – ціла частина від x $]x[$ - округлення до найближчого більшого цілого. На першому кроці алгоритму замість змінних m_1 і n_1 підставляються реальні значення m і n, на подальших, якщо це можливо, – відповідні значення попереднього кроку з правої частини співвідношення (1).

Диз'юнктивний вираз вигляду:

$$K'(m_1\text{-}p,[n_1/2])\ ?K'(p,]n_1/2[) \qquad (2)$$

де $0<p<m_1$, $n1>m_1$, $m_1-p\leq[n_1/2]$, $p\leq]n_1/2[$, піддається спеціальним перетворенням і зрештою може бути записаний у вигляді однієї булевої реберної функції, тобто відповідна множина *K'* складатиметься з одного елементу.

Відзначимо, що у виразах (1) і (2) операцію диз'юнкції слід відносити не до самих множин, а до пар елементів (функцій) різних множин виду *K'*.

Множина $K'(m_1,n_1)$, де $m_1=n_1$, не потребує представлення у вигляді (2) і подальшого перетворення: воно складається тільки з однієї функції.

Зрозуміло, що розглядати рух в глибину на наступному кроці рекурсії по (1) доцільно тільки для диз'юнкцій виду $K'(m_1,[n_1/2])? K'(0,]n_1/2[)$ або $K'(0,[n_1/2])?K'(m_1,]n_1/2[)$, тобто при граничних значеннях змінної *i*, тому що тільки в цьому випадку нам необхідний подальший рекурсивний спуск для визначення *K'(m,n).* Відзначимо, що множина $K'(m_1,n_1)$ порожня при $m_1=0$, тому надалі вона буде опускатися.

Після виконання всіх перетворень вигляду (1) і (2), будуть отримані всі множини типу *K'*, для яких не потрібні або неможливі подальші перетворення. Для отримання шуканої *K'(m,n)* виконуємо об'єднання елементів цих множин.

Множини типу *K'* для яких не потрібні або неможливі подальші перетворення складаються з однієї функції і відповідають вершинам дерева ієрархії реберних функцій. Для моделі *K*(*4,13*) таких функцій буде 10 (рис. 4).

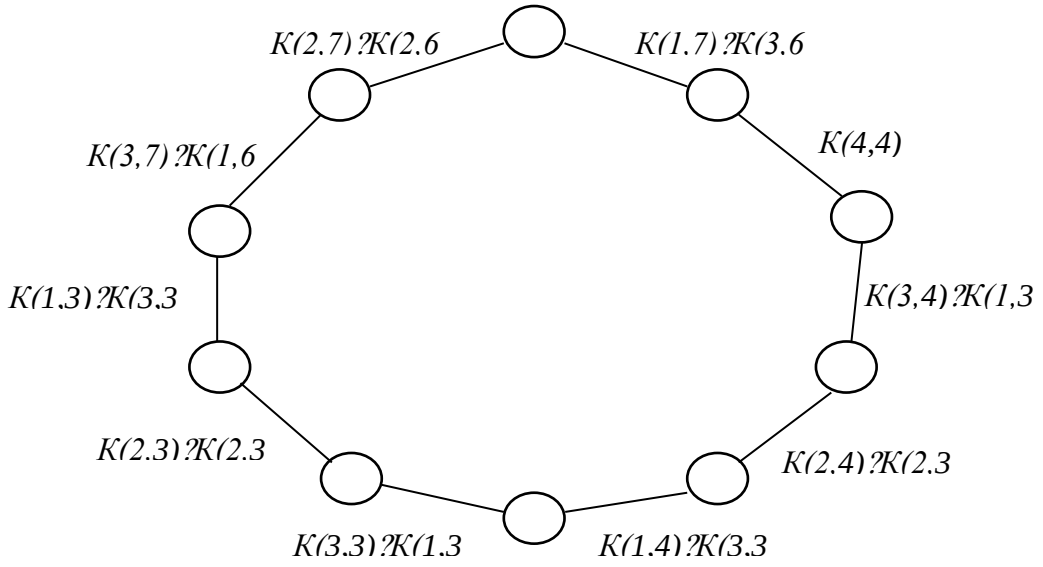


Рис. 4. GL-модель для *K*(*4,13*)

ВБС *K*(*4,13*) складається з 13 модулів (процесорів), і стійка до відмови не більше 4-х з них. Диз'юнкції приписані ребрам графу, який зображений на рис. 4 насправді є реберними функціями, побудова яких розглянута не буде. Головне розуміти, що ці функції залежать від 13 змінних, кожна з яких відповідає стану певного модуля системи. Диз'юнкції зображені на рис. 4 для наочного співвідношення вершин дерева ієрархії ребрам графу.

Таким чином, розглянутий вище алгоритм дозволяє побудувати GL-модель для довільної базової відмовостійкої багатопроцесорної системи. Кількість ребер у графі моделі будемо позначати літерою *r*.

Варто відзначити декілька важливих властивостей мінімізованої GL-моделі, побудованої за вище описаним алгоритмом:

- Кількість *r* ребер мінімізованої GL-моделі визначається за формулою:

$$r = n - m + 1 \qquad (3)$$

Доведення цієї формули можна знайти в іншій літературі по GL-моделям, і приводитись в даній роботі не буде. Можна відзначити

справедливість цієї формули для розглянутої вище ВБС *K*(*4,13*). Справді 13 – 4 + 1 = 10.

- Мінімізована GL-модель втрачає мінімальну кількість ребер при появі вектора стану системи з *m+1* нулем, тобто рівно 2 ребра. Для моделі *K*(*4,13*) це означає, що при появі 5 відмов з графу вилучиться рівно 2 ребра, тобто рівно 2 реберні функції приймуть значення 0. Інші 8 реберних функцій будуть мати значення 1.

1.2 Перетворення GL-моделі

В попередньому підрозділі було визначено поняття базової *k* – ВБС – такої ВБС, яка містить *n* модулів і стає непрацездатною при відмові не менше ніж *k+1* модулів, за умови *k<n*

Проте, в реальних ВБС часто буває, що система продовжує функціонувати і при виході з ладу деяких модулів, де число останніх перевищує величину m. Наприклад, ВБС може реконфігуруватися в більшості випадків 2-кратних відмов її модулів, проте на певній сукупності векторів стану модулів системи, що містять два нулі, вона відмовляє, залишаючись в той же час працездатної на деяких сукупностях векторів станів, що мають 3 або навіть 4 нулі. Подібні вимоги диктуються конкретними міркуваннями, архітектурою, конфігурацією системи, характеристиками модулів, їх можливостями виконання функцій інших модулів в процесі реконфігурації і іншими чинниками. Іншими словами, дослідження ВБС не можна обмежувати лише базовими системами. Для аналізу поведінки таких систем потрібні складніші моделі.

В даному розділі буде розглянутий процес побудови небазових ВБС.

Нехай є базова система *K(m,n)* і її GL-модель. Потрібно побудувати GL-модель для системи, яка має властивості базової Knm (тобто вона стійка

до відмов не більш m своїх модулів), і залишається працездатною ще на векторі станів ?:

$$x_1\ x_2\ x_3\ \dots\ x_m\quad x_{m+1}\ x_{m+2}\ \dots\ x_n$$
$$0\ \ 0\ \ 0\ \dots\ 0\quad\ 0\quad\ 1\quad\dots\ 1$$

Тут, x_i – модуль системи, який може бути у стані 1- справний, або – 0 - несправний. Іншими словами, наша модернізована GL–модель не повинна втрачати зв'язності при появі вектора станів ?.

Побудуємо базову GL-модель $K(m,n)$. Відзначаємо, що при появі вектора станів ? не менше двох функцій, що приписуються ребрам циклічного графу, приймуть нульове значення. У графі GL-моделі зникнуть два ребра, і він втратить зв'язність, тобто GL-модель $K(m,n)$ не адекватна системі, яку ми повинні побудувати. Нехай для конкретності це будуть рівно дві функції: f_1 та f_2.

Справедливо наступне: для збереження адекватності GL-моделі на векторі ? достатньо ребру з функцією f_1 приписати функцію $f_1^*=f_1 ? H$, де H – конституента одиниці, відповідна набору ?:

$$H = \bar{x}_1 \cdot \bar{x}_2 \cdot \ldots \cdot \bar{x}_{m+1} x_{m+2} \cdot \ldots \cdot x_n$$

Дійсно, ребро графу з новою функцією f_1^* при появі вектора ? не зникає ($f_1^*=1$), отже, зв'язність графа зберігається. При появі інших векторів стану функція f_1^* приймає такі самі значення, як функція f_1, і, відповідно, адекватність GL-моделі не порушується. Якщо ж при появі вектора ? на початковій базовій GL-моделі зникає більше двох ребер одночасно, достатньо всі функції цих ребер, окрім однієї, змінити таким же чином, тобто додати ту ж конституенту одиниці H. Зрозуміло, що в цьому випадку саме на наборі ? зв'язність графа не порушиться (випадає лише одне ребро).

Тут слід зазначити, що одне і те саме ребро може пропадати при появі багатьох, наприклад *s*, векторів, з множини, яка задана. Якщо *s* відповідних конституент одиниці додати до функції f_1, яка присвоєна цьому ребру, то отриманий вираз

$$f_1^* = f_i \; ? H_1 ? H_2 \; ? \ldots \; ? H_s$$

може бути мінімізований будь-яким з відомих методів мінімізації форм представлення булевих функцій, що ніяк не відіб'ється на адекватності GL-моделі. Зрозуміло, що вибір конституент одиниці, які приписується тим або іншим ребрам, може суттєво впливати на кінцеву форму представлення реберних функцій.

Слід відзначити, що даний підхід є універсальним в тому сенсі, що дозволяє будувати GL-моделі як для систем зі зниженим ступенем відмовостійкості, так і для систем з підвищеним ступенем відмовостійкості по відношенню до базових GL-моделей.

Додаткове ребро *r* з функцією *f*, що з’єднує дві які-небудь вершини GL–моделі, змінює останню тільки в одному напрямі: нова GL-модель адекватна системі з більш високим ступенем відмовостійкості.

Насправді, функція *f* може приймати два значення: 0 і 1. Нульове значення функції, тобто відсутність додаткового ребра, не змінює зв'язності початкового графа. Одиничне значення функції *f* підвищує «ступінь» зв'язності, оскільки з'являються нові шляхи на графі.

Запишемо основні постулати проведення додаткових ребер в графі для побудови нової GL-моделі:

1. Якщо при появі заданого вектора станів ? випадають ребра a_i та a_j, то додаткове ребро *r* повинно проходити між ними.

Очевидно, що якщо додаткове ребро r не буде проходити між ребрами a_i та a_j, то порушення зв'язності графа GL-моделі відбудеться незалежно від функції, що приписується ребру r.

2. Якщо при появі заданого вектора станів ? випадає декілька ребер a_1, a_2 … a_k, то, ребра r_1 … r_L, що вводяться в базову GL-модель, повинні бути проведені так, щоб між двома найближчими вершинами,які інцидентні двом довільним додатковим ребрам, знаходилося не більш ніж одне ребро a_i ?*?*a_1, a_2 ... a_k*?* початкового графу.

Інакше два ребра a_i , a_j ?*?*a_1, a_2 ... a_k*?*, що знаходяться між вказаними вершинами, утворюють ізольовану частину графу при появі вектора ?, і, таким чином, зв'язність графа порушиться.

3. Для збереження зв'язності графу при появі вектора ? достатньо ребру r (або ребрам r_1 … r_L), побудованому вказаним чином, приписати функцію H (тобто, ту ж конституенту одиниці, що і раніше в 1.4.1).

Дійсно, таке ребро r (або ребра r_1 … r_L з однією і тією ж функцією) запобігає втраті зв'язності графу, оскільки забезпечує шлях на графі між ділянками кільця, роз'єднаними зниклими ребрами. З іншого боку, воно не змінює GL-модель на інших векторах, оскільки відсутнє при появі всіх векторів станів, окрім ? ($H = 1$ тільки на одному наборі).

Нехай в базовій GL-моделі для вектора ? проведено додаткове ребро r_1 з функцією H_1. Нехай існує інший вектор стану β, для якого необхідно вирішити таку ж задачу. Тоді, якщо ребра a_i та a_j, які випадають при появі вектора β, лежать по різні сторони від ребра r_1, то, достатньо, для збереження адекватності GL-моделі приписати ребру r_1 функцію $f=H_1?H_2$
де H_2 - конституента одиниці, яка відповідає вектору β. Якщо ребра a_i та a_j лежать

по одну сторону від ребра r_1, то для вирішення задачі необхідне інше додаткове r_2, яке слід проводити, керуючись тими ж правилами.

Іншими словами, кожному додатковому ребру приводиться у відповідність функція, яка є диз'юнкцією конституент тих векторів стану модулів системи, для яких саме це додаткове ребро на GL-моделі повинно запобігати втраті зв'язності.

Реберна функція фактично є частково визначеною. Зокрема, її значення на векторах станів, поява яких призводить до зникнення ребер базової GL-моделі, що знаходяться по один бік від ребра r_1, можуть бути встановлені довільно, оскільки не впливають на функціонування GL-моделі.

Розглянемо наступний приклад.

Побудуємо GL-модель 4-модульної системи, яка стійка до всіх одиночних відмов і, крім того, до відмов груп модулів {1,3} і {2,3}.

Побудована базова модель *K(1,4)* зображена на рис. 5.

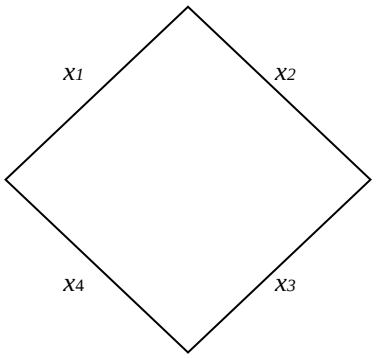


Рис. 5. Графологічна модель *K(1,4)*

Для даного прикладу достатньо одного внутрішнього ребра. Функція f визначається легко:

$$f = \overline{x}_1 \overline{x}_3 x_2 x_4 \vee \overline{x}_2 \overline{x}_3 x_1 x_4$$

Враховуючи також, що функція *f* може приймати довільні значення на всіх наборах виду 00*xx* і *xx*00 (тут *x* означає будь-яке значення 0 або 1), та використовуючи відомі методи мінімізації не повністю визначених булевих функцій, отримуємо:

$$f = \overline{x}_3 \cdot (\overline{x}_1 \vee \overline{x}_2)$$

В даному випадку можна відзначити, що функція *f* не визначена також на наборах 1110, 1101, 1011, 0111, тобто на векторах станів системи, коли з ладу виходить не більш ніж один модуль. Це дозволяє спростити її ще більше:

$$f = \overline{x}_3$$

Адекватна прикладу GL-модель зображена на рис. 6.

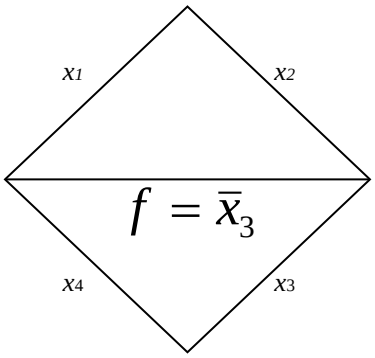


Рис. 6. Небазова GL-модель

Варто відзначити, що кожне, побудоване згідно правилам, описаним вище, нове додаткове ребро, змінює GL-модель лише для одного вектора ? станів модулів ВБС, залишаючи всі властивості і характеристики моделі незмінними для всіх інших векторів. Додаткових ребер може бути декілька і це залежить від кількості векторів.

1.3 Додаткові ребра у GL –моделях

Розглянемо конкретну базову GL-модель *K(m,n),* представлену у вигляді циклічного графу R. Позначимо кількість ребер графа як *r*, а множину його ребер як $A=\{a_1,a_2,a_3 \dots a_r\}$.

Визначимо $W=\{w_1,w_2,w_3 \dots w_n\}$, як множину визначених векторів (наборів) станів модулів (вхідних змінних $x_1,x_2 \dots x_n$), в яких кратність відмови перевищує *m*, але в той же час система продовжує функціонувати.

Визначимо величину t_i як кількість ребер графу R, які зникають на наборі w_i, а також максимальну кількість ребер, які одночасно зникають $t_{max}=max(t_i)$.

Кожен набір з *W* визначає сукупність ребер графу R, на яких функції приймають нульове значення. Множину таких сукупностей ребер позначимо $Q=\{q_1,q_2, \dots q_n\}$, $q_k=\{a_{1k},a_{2k, \dots}\}$.

Позначимо множину всіх пар ребер, які входять в q_i, як q_i*.

Побудуємо множину $P = q_1*? q_2*? \dots ? q_n*$. Множина *P* складається з усіх пар ребер, що входять в елементи множини Q, що означає, що при появі будь-якого вектору з множини *W*, пари ребер, що виключаться з графу, присутні в множині *P*. Очевидно, що для випадку $t_{max}=2$ $P=Q$.

Для збереження адекватності GL-моделі на множині векторів *W* необхідно провести у графі R додаткові ребра. Задача оптимізації їх проведення розглянута в [2], і базується на понятті S-підмножини.

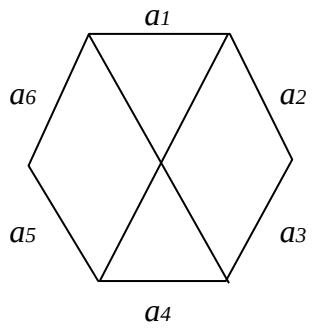
Дві або більше непустих непересічних підмножини основних ребер визначимо як S-підмножини, якщо зв'язність графу зберігається при втраті будь-якої пари ребер, що належать різним підмножинам, і якщо зв'язність графу порушується при втраті будь-якої пари ребер, що належать будь-якій з підмножин, що містить більш ніж одне ребро.

Розглянемо приклад. Проведемо додаткове ребро (для простоти вважаємо його функцію *f=1*), яке розділяє множину основних ребер *A* на дві

непересічні підмножини A_1, A_2, що мають наступну властивість: зв'язність графа зберігається при втраті будь-якої пари ребер $\{a_i,a_j\}$ $a_i ?A_1$, $a_j ?A_2$. З іншого боку, зв'язність графа порушиться при втраті будь-якої пари з A_1 або з A_2. В даному випадку A_1 і A_2 є S-підмножинами.

Кожне нововведене додаткове ребро розділяє вже отримані підмножини на менші S-підмножини. Таким чином, можливе розбиття множини A на довільну кількість S-підмножин, що не перевищує кількість основних ребер. Через S-підмножини легко отримати множину «захищених» пар ребер (під «захищеною» парою розуміємо пару основних ребер, при видаленні яких зв'язність графа зберігається).

На рис. 7 приведений приклад графу розширеної GL-моделі з двома додатковими ребрами, що розділяють множину основних ребер на 4 S-підмножини.



Кількість ребер циклічного графу: $r=6$
Кількість S-підмножин: $?=4$
S-підмножини: $S=\{\{a_1\},\{a_2,a_3\},\{a_4\},\{a_5,a_6\}\}$
Кількість додаткових ребер: $e=2$
Множина «захищених» пар: $\{a_1,a_2\},\{a_1,a_3\}, \{a_1,a_4\}, \{a_1,a_5\},\{a_1,a_6\},\{a_2,a_4\},\{a_3,a_4\},\{a_2,a_5\},\{a_2,a_6\},\{a_3,a_4\},\{a_3,a_5\},\{a_3,a_6\}, \{a_4,a_5\},\{a_4,a_6\}$

Рис. 7. Розбиття графу на S-підмножини

Для забезпечення зв'язності на заданих наборах при $t_{max}=2$ можна побудувати граф R*, упорядкувавши ребра графу R і провівши додаткові ребра, тим самим розділивши основні ребра на S-підмножини так, щоб вершини кожної з пар знаходилися в різних S-підмножинах. Функції, приписані додатковим ребрам, повинні формуватися так, щоб зв'язність графу, при появі інших наборів, залишалася відповідною базовій моделі.

Побудуємо граф V: $V=\{A,P\}$. Іншими словами, як множину вершин візьмемо множину ребер графу R: $A=\{a_1,a_2,a_3 \ldots a_R\}$, а як ребра – множину пар

P. Це означає, що вершини a_i та a_j в графі V будуть суміжними, якщо необхідно забезпечити зв'язність графа R за відсутності ребер a_i та a_j.

Очевидно, що визначення S-підмножин ребер графу R еквівалентно визначенню внутрішньо стійких множин вершин графа *V* (внутрішньо стійкою множиною вершин називається підмножина вершин графу не зв'язаних між собою). Таким чином, знайшовши розбиття множини вершин графу *V* на мінімальне число внутрішньо стійких множин, ми знаходимо шукані S-підмножини ребер графу R.

Задача пошуку S-підмножин графу R тепер може бути зведена до розфарбовування вершин графу *V*. Вершини, що отримали один колір, належать одній внутрішньо стійкій множині. Таким чином, мінімальна кількість фарб, необхідних для розфарбування графу *V* визначає оптимальну кількість S-підмножин, які в свою чергу визначають мінімальну кількість додаткових ребер, які необхідно провести в GL-моделі для збереження її адекватності на наборах векторів *W*. Нагадаємо, що мінімальна кількість внутрішньо стійких множин графу рівна його хроматичному числу *?*.

В [2] доведена формула визначення оптимальної кількості додаткових ребер для $t_{max}=2$:

$$e=[(?+1)/2]$$

де *?* хроматичне число *V* графу, або кількість S-підмножин, що те саме.

Підсумувавши вище сказане, можна привести алгоритм оптимальної побудови розширеної GL-моделі для випадку $t_{max}=2$:

- Побудувати базову GL-модель *K(m,n)* за алгоритмом описаним в 1.2. Створена модель є мінімізованою, що означає, що будь-який вектор станів модулів системи з m+1 нулем призводить до виключення з графу рівно двох ребер, що задовольняє умові $t_{max}=2$.

- Як вхідні дані отримати множину $W=\{w_1,w_2,w_3...w_n\}$ – множину визначених векторів (наборів) станів модулів (вхідних змінних $x_1,x_2...x_n$), в яких кратність відмови перевищує m, але в той же час система продовжує функціонувати. Для забезпечення відповідності умові $t_{max}=2$, поява будь-якого вектора w_i з множини W повинна призводити до виключення з графу R не більше двох ребер.
- Отримати множину P, як множину таких сукупностей ребер, які випадають з графу при появі вектора з множини W. За умови $t_{max}=2$ потужність множини P дорівнює потужності множини W.
- Побудувати граф V: $V=\{A,P\}$.
- Розфарбувавши граф V, отримати мінімальну кількість S-підмножин, необхідних для побудови розширеної GL-моделі.
- Згідно з [2] провести додаткові ребра в базовій GL-моделі.

1.4 Границі числа додаткових ребер

На різних етапах розробки, аналізу і трансформації відмовостійких багатомодульних систем (ВБС), особливо систем управління складними об'єктами, GL-модель, яка моделює поведінку такої системи, іноді виходить достатньо складною. У зв'язку з цим практичний інтерес представляє розрахунок параметрів моделі на перших етапах проектування ВБС, а також визначення складності трансформації вже створеної моделі при перетворенні самої системи. Складність моделі впливає на час виконання статистичних експериментів з нею і, отже, на точність розрахунку надійності ВБС (через кількість самих експериментів).

Як було сказано в попередніх розділах, реальні ВБС рідко задовольняють критеріям базових, що призводить до необхідності розробки і

моделювання більш складних ВБС. В підрозділі 1.3 був розглянутий оптимальний метод проведення внутрішніх ребер в перетвореній GL-моделі з метою забезпечення адекватності її поведінки в потоці відмов при появі векторів стану системи з більш ніж *m* нулем. Така розширена GL-модель забезпечує збереження зв'язності графу при випаданні з нього більш ніж 1 ребра шляхом проведення додаткових внутрішніх ребер.

Будемо розглядати випадок, коли з графу випадає лише пара основних ребер.

Позначимо множину пар ребер, при випаданні яких потрібно забезпечити зв'язність графу, як *L*, а його потужність - через *l*. Надалі для скорочення викладень збереження зв'язності графу при зникненні пари ребер (з множини *L*) називатимемо блокуванням відповідного вектора, або просто блокуванням.

Відповідно до алгоритму, який описаний вище, ребра графа GL-моделі можна розбити на непусті непересічних S-підмножини (дві або більше): у одній S-підмножині присутні тільки такі ребра, вибірка по два яких не співпадає ні з одним елементом з множини L. Дуже важливо виконати розбиття графу на S-підмножини таким чином, щоб їх кількість була мінімальною. Далі на графі молюються додаткові ребра відповідно до розбиття графу на S-підмножини. Фактично розбиття множини ребер графу на S-підмножини визначає кількість і місце проведення додаткових ребер, які слугують збереженню зв'язності графа при зникненні тієї чи іншої пари ребер. Оскільки проведення додаткових ребер фактично визначається розбиттям графа на S-підмножини, вважатимемо, що S-підмножини блокують певні пари ребер з множини *L*. Позначимо *p* - кількість S-підмножин.

Сформулюємо задачу: визначити межі кількості додаткових ребер (або S-підмножин), що вводяться для забезпечення адекватності моделі (при її

трансформації), якщо відома потужність множини W, і, з другого боку, визначити межі, в яких може знаходитися величина потужності множини W, поява векторів з якої не приведить до втрати зв'язності графу моделі, якщо задано число додаткових ребер (або S-підмножини).

Переформулюємо задачу трохи в іншому вигляді:

- визначити межі p (кількість S-підмножин), виходячи із заданих r (кількість основних ребер моделі) і l (потужність множини L).
- визначити межі для l при заданих p і r.

Більш менш точна оцінка цих меж дозволяє розробнику ВБС розрахувати параметри моделі на ранніх етапах її проектування, які впливають на час виконання одного статистичного експерименту з нею і, як відмічалось вище, на точність розрахунку ймовірності безвідмовної роботи ВБС.

Визначення: Верхня межа $p_{max}(l)$ – це така кількість S-підмножин, яка здатна блокувати будь-який набір елементів множини L, потужністю в l, виконуючи умову мінімальності кількості S-підмножин. $p_{min}(l)$ – це мінімальна кількість S-підмножин, що забезпечує можливість блокування хоча б одного набору з l пар ребер, причому додавання хоча б одного елементу в множину L однозначно призвело б до розбиття на більше, ніж $p_{min}(l)$ кількості S-підмножин.

Визначення: $l_{min}(p)$ – таке мінімальне значення l, що для блокування елементів з множини L потрібне оптимальне розбиття множини ребер графу на p S-підмножин. Тобто це така комбінація з l пар, видалення хоча б одного елементу з множини L призвело б до розбиття на менш ніж p S-підмножин (з урахуванням вимоги мінімальності до їх кількості). $l_{max}(p)$ – це максимальна кількість пар ребер, для блокування яких достатньо розбиття графу на p S-підмножин.

Оцінка цих границь розглядається в іншій літературі по GL-моделям. Нас буде цікавити границя $l_{max}(p)$, причому у випадку, коли множина L формується з пар ребер, які випадають при появі заданих векторів з $m+1$ нулем.

2 Оцінка складності моделі

В розділі 1 була представлена GL-модель, як один з варіантів моделей, яка призначена для розрахунку надійності ВБС. Були описані алгоритми побудови базової GL-моделі, та способи перетворення її до розширеної (небазової).

Однією з важливих задач на ранніх етапах побудови GL-моделі є оцінка її складності. Складність моделі впливає на час роботи кожного статистичного експерименту з нею, що в свою чергу впливає на точність розрахунку надійності ВБС. У випадку, коли точність розрахунку надійності ВБС не задовольняє вимогам, потрібно шукати способи зменшення складності ВБС (і моделі), або збільшувати кількість тестів. Знаючи ці вимоги заздалегідь можна оцінити доцільність побудови тієї чи іншої GL-моделі для заданої ВБС.

Розглянемо базову GL-модель $K(m,n)$. Мінімізована за алгоритмом, нагаданим в 1, модель буде мати $n–m+1$ ребро і при появі вектора з $m+1$ нулем буде втрачати рівно 2 ребра. Нехай на моделі необхідно відобразити спроможність ВБС залишатися роботоздатною на заданому наборі векторів стану системи з $m+1$ нулем. Визначимо $W=\{w_1,w_2,w_3...w_n\}$, як множину таких векторів, а L – як множину усіх пар ребер, які випадають з графу при появі векторів з W. Зрозуміло, що кількість пар ребер з множини L не перевищує кількості векторів множини W.

В розділі 1 був описаний процес перетворення базової GL-моделі до небазової шляхом проведення додаткових ребер. При умові $t_{max}=2$ існує метод оптимального проведення додаткових ребер, який заснований на розфарбуванні V графу. Нагадаємо, що V граф – це такий граф, вершинами якого є ребра початкового графу GL-моделі, а кожному ребру відповідає

пара з множини *L*. Розфарбування *V* графу дозволяє отримати S-підмножини, на основі яких проводяться додаткові ребра в графі.

Однією з складових оцінки складності перетворення GL-моделі з базової до розширеної є верхня межа $l_{max}(p)$ – це максимальна кількість пар ребер, для блокування яких достатньо розбиття графу на *p* S-підмножин. Тобто задачею знаходження $l_{max}(p)$ є відповідь на питання: скільки максимально пар ребер можна заблокувати, провівши в графі *e* додаткових ребер, або розбивши граф на *p* S-підмножин. Насправді розбиття графу на *p* S-підмножин визначає оптимальну кількість *e* додаткових ребер:

$$e=]p/2[$$

Задачу розбиття графу на *p* S-підмножин можна звести до задачі розфарбування *V* графу в *p* кольорів. Тепер задачу можна переформулювати наступним чином: знайти таке розташування *p* фарб у *V* графі, щоб в ньому можна було провести максимальне число ребер. По суті в даному контексті під максимальним числом мається на увазі максимальне серед інших варіантів розташування *p* фарб у *V* графі.

2.1 Дерево ієрархії і пари ребер, що виключаються з графу

В якості *V* графу для розфарбування можна взяти дерево ієрархії, описане в 1.1 (рис. 3). На цьому дереві, так само як і на *V* графі, вершинами позначаються ребра базової GL-моделі. Проте на дереві ієрархії існують вузли, які не відповідають ребрам GL-моделі і є зайвими для процесу розфарбування графу, тому трансформуємо його, видаливши з нього усі вузли, а дуги, які з них виходили, припишемо сусіднім до вузлів вершинам. Приклад такого дерева зображений на рис. 8. Надалі під деревом ієрархії будемо розуміти саме це дерево.

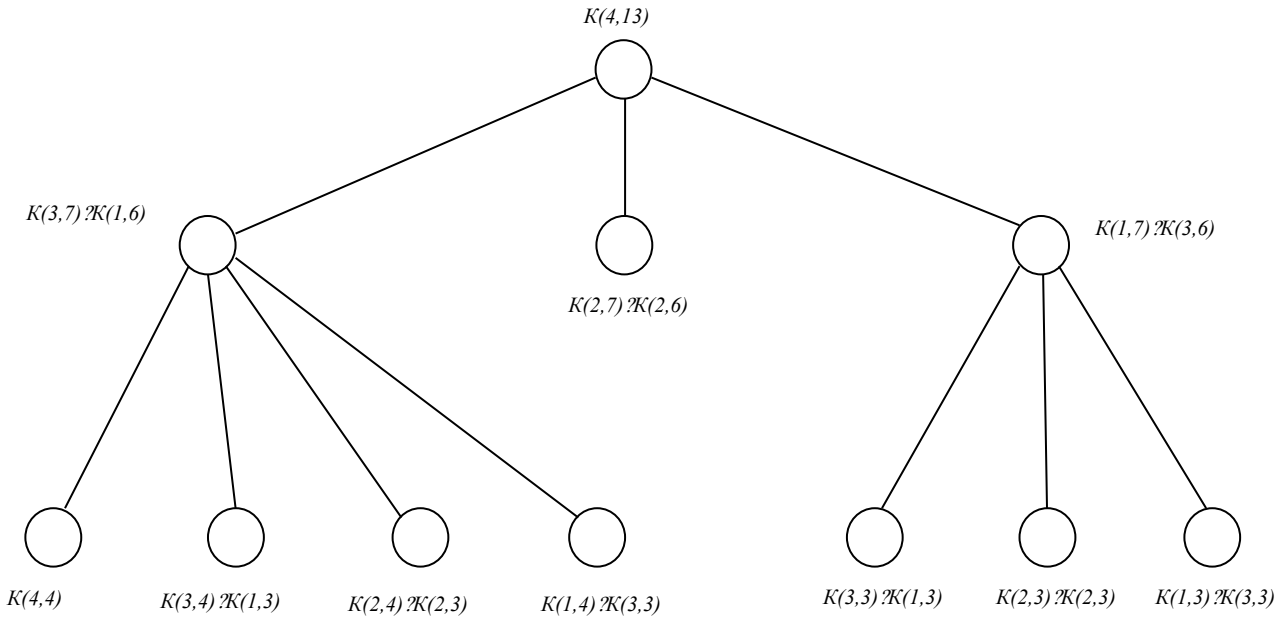


Рис. 8. Модифіковане дерево ієрархії

Наведемо твердження, доведене в іншій літературі:

Твердження 1:

Два ребра GL-моделі базової ВБС K(m,n) можуть виключатися з графу моделі одночасно при появі будь-якого вектора стану системи, що містить m+1 нульову компоненту, тоді і тільки тоді, коли вони відповідають двом вершинам дерева ієрархії реберних функцій моделі, які:

а) або розташовані на одній гілці дерева,

б) або є сусідами.

Сусідніми будемо вважати ребра, що виходять з одного вузла (на дереві вони розташовані поряд).

Дане твердження накладає певні обмеження на проведення ребер у дереві. При розфарбуванні дерева треба мати навазі, що ребра можна проводити лише між певними, визначеними в твердженні, вершинами.

Таким чином задача формулюється наступним чином: знайти алгоритм такого розташування *p* фарб у дереві ієрархії, яке б дозволяло провести в

ньому максимальну кількість ребер, таких які б задовольняли твердженню 1. Кількість ребер позначатимемо $l_{max}(p)$.

2.2 Алгоритм

Наша задача розфарбування дерева трохи відрізняється від класичної задачі пошуку хроматичного числа графу. При пошуку хроматичного числа, граф має бути повністю визначений, тобто мати певну визначену кількість вершин і розташування ребер між ними. Після цього, вершинам графу приписуються фарби і шукається їх мінімальна кількість за певним алгоритмом.

Процедура розфарбування графу (дерева) для вирішення задачі, що сформульована, має дещо інший характер. Задана лише кількість вершин графу, і треба виконати їх розфарбування, після чого підрахувати кількість ребер, які можливо в ньому провести. Нагадаємо, що на проведення ребер накладаються обмеження визначені у твердженні 1, а саме: вершини, інцидентні ребру, повинні розташовуватись на одній гілці дерева, або бути сусідами, окрім того, що вершини інцидентні ребру мають бути пофарбовані в різні кольори.

Одним з методів вирішення даної задачі (як завжди) є повний пошук розфарбування дерева, під яким мається на увазі перебір усіх варіантів розташування фарб у дереві, і підрахунок для кожного з варіантів кількості ребер, які можна провести при заданому розташуванні фарб. І тільки після цього знаходиться максимальна кількість ребер. Це і буде $l_{max}(p)$.

Однією з важливих вимог до алгоритму пошуку є те, що він має бути безповторним, тобто в процесі пошуку алгоритм не повинен більше одного разу аналізувати одне і те саме розташування фарб, або еквівалентне розташування.

Вхідні дані для нашого алгоритму пошуку: кількість фарб *p* та кількість *r* вершин дерева (або, що теж саме, кількість ребер побудованої GL-моделі).

Фактично завданням першої частини алгоритму є генерування векторів фарб вершин дерева. Під вектором розуміється вектор довжиною *r*, елементами якого є номери фарб, в які можуть бути пофарбовані відповідні вершини дерева.

Таким чином, алгоритм пошуку повного розфарбування можна представити в наступному вигляді:

1. $l_{max}(p) := 0$.
2. Генерування чергового вектора фарб, перехід до пункту 3. Якщо були розглянуті усі вектори – перехід до пункту 4.
3. Підрахунок кількості *l* ребер, які можна провести в дереві, при заданому розташуванні фарб. Якщо $l > l_{max}(p)$, то $l_{max}(p) := l$. Перехід до пункту 2.
4. Кінець.

Цікавим тут є алгоритм безповторного генерування вектора фарб.

Для початку необхідно визначити усі комбінації розташування фарб по вершинах дерева. Наведемо приклад для $r = 9$, $p = 3$.

№	Кількість вершин, які пофарбовані у відповідний колір		
1	1	1	7
2	1	2	6
3	1	3	5
4	1	4	4
5	2	2	5
6	2	3	4
7	3	3	3

Табл. 1. Комбінації розташування фарб по вершинам дерева

В першому стовпчику розташована кількість вершин, пофарбованих у перший колір, в другому у другий і т.д. Очевидно, що в кожному рядку сума чисел дорівнює кількості вершин дерева, в даному випадку 9. Ці комбінації не мають повторюватись, тобто з точністю до перестановки стовпчиків не повинно існувати двох однакових рядків,. Як видно з таблиці, усього таких комбінацій для даного прикладу буде 7. Позначимо множину таких комбінацій як $RP = \{rp_1, rp_2, \dots, rp_s\}$, де $rp_i = [r_{i1}, r_{i2}, \dots, r_{ip}]$ – вектор кількості фарб.

Наступним кроком алгоритму є безповторне генерування векторів фарб, для кожного елементу множини *RP*. Скажімо, для rp_4 = [1, 4, 4] можна пофарбувати одну вершину в перший колір, чотири вершини у другий колір і чотири вершини у третій колір. Одним з векторів фарб для rp_4 може бути наступний:

3	2	3	2	2	3	2	1	3

Рис. 9. Вектор фарб для rp_4

Нагадаємо, що *i-м* елементом вектора фарб є номер фарби, в який пофарбована *i-а* вершина графу. Справді, в колір 1 пофарбована одна вершина, в колір 2 – чотири і в колір 3 – чотири. Завданням пошуку є генерація усіх можливих векторів фарб для усіх елементів множини *RP*.

Безповторності можна досягти розташовуючи фарби послідовно. Нехай в перший колір повинно бути пофарбовано r_1 ребер. Очевидно, що кількість варіантів розташування першої фарби в дереві буде:

$$C_r^{r1}$$

Після розфарбування першим кольором, залишається розташувати *p–1* фарбу серед $r–r_1$ вершин дерева. Тобто алгоритм розташування фарб є рекурсивним. Тоді повну кількість векторів фарб для кожного елементу *RP* можна визначити за наступною формулою:

$$C_r^{r1} C_{r-r1}^{r2} C_{r-r1-r2}^{r3} \dots C_{rp}^{rp} = \prod_{i=1}^{p} C_{r-\sum_{j=1}^{i-1} ri}^{ri}$$

Наприклад для *rp4* кількість векторів фарб дорівнює $C_9^1 C_8^4 C_4^4 = 630$.

Проте, при генерації векторів вище описаним методом, можлива ситуація їх дублювання, а точніше, можуть з'явитися еквівалентні вектори. Еквівалентними є однакові вектори, з точністю до перестановки фарб.

3	2	3	2	2	3	2	1	3

2	3	2	3	3	2	3	1	2

Рис. 10. Приклад еквівалентних векторів

Вектори, зображені на рис. 10, є еквівалентними. Різниця між ними полягає лише в перестановці другої і третьої фарб місцями. Очевидно, для даної задачі не є важливим сам колір, тому розфарбування графу за такими еквівалентними векторами дасть одну і ту саму кількість ребер. Тому в алгоритмі генерації векторів це слід враховувати і уникати такого дублювання.

Зазначимо, що ситуація з появою еквівалентних векторів може трапитись тільки у випадку, коли у векторі *rp* двом або більше кольорам, буде приписана однакова кількість ребер. В наведеному вище прикладі, вектор *rp4* = [1, 4, 4], що і призвело до появи еквівалентних векторів.

Так як вектори *rp* є унікальними, то жодні 2 вектори фарб які відповідають різним векторам *rp* не можуть бути однаковими або еквівалентними. Отже ситуація еквівалентності векторів може трапитись лише в рамках одного вектора *rp*.

Варто зазначити, що в процесі генерування векторів фарб для певного вектора *rp*, кожному кольору приписана незмінна кількість вершин, які

мають бути пофарбовані в цей колір. Це випливає з рекурсивності алгоритму генерування векторів. Скажімо для $rp_4 = [1, 4, 4]$ в перший колір буде пофарбована лише одна вершина, тому, зображені на рис. 11 еквівалентні вектори одночасно з'явитись в процесі генерування не можуть.

3	2	3	2	2	3	2	1	3

3	1	3	1	1	3	1	2	1

Рис. 11

В процесі пошуку буде згенерований лише перший вектор (рис. 11), в якому в перший колір пофарбована лише 1 вершина.

З усього вище сказаного і випливає, що еквівалентні вектори можуть з'явитися лише у випадку коли у векторі *rp* двом або більше кольорам, буде приписана однакова кількість ребер (рис. 10).

Нехай певний вектор *rp* з множини *RP* містить *u* фарб, в які необхідно розфарбувати по *v* ребер графу (рис. 12).

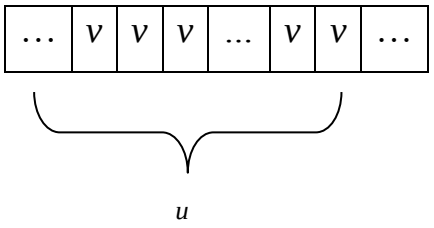


Рис. 12

В процесі генерування векторів фарб для даного вектора *rp* будуть траплятися ситуації дублювання векторів. Кожна з *u* фарб може бути заміщена будь-якою іншою з *u-1* фарб, що призведе до появи еквівалентних векторів. Всього кількість еквівалентних векторів буде дорівнювати

$$u!$$

як кількість перестановок *u* фарб. Тобто, для кожного вектора фарб, згенерованого алгоритмом, буде існувати *u! – 1* еквівалентних. Таким чином, уникнувши дублювання векторів, можна зменшити перебір у *u!* разів.

Розглянемо конкретний приклад. Нехай *rp = [1, 3, 3, 3, 3]*, звідки *v = 3*, *u = 4, p = 5, r = 13.* Кількість згенерованих векторів (включаючи еквівалентні) буде дорівнювати $C_{13}^{1}C_{12}^{3}C_{9}^{3}C_{6}^{3}C_{3}^{3} = 4804800$. Уникнувши дублювання векторів ми зможемо зменшити цю кількість у *4! = 24* рази, тобто до 200200.

Спробуємо вирішити цю задачу, залишившись в межах запропонованого рекурсивного алгоритму.

На найвищому рівні рекурсії, вибирається 1 ребро для його розфарбування у *1-й* колір. Очевидно, що кількість варіантів розфарбування на цьому рівні буде дорівнювати $C_{13}^{1} = 13$. На другому рівні необхідно розфарбувати 3 ребра (з 12) у другий колір. Проте, перебір усіх можливих варіантів розфарбування на цьому рівні ($C_{12}^{3} = 220$) неодмінно призведе до появи еквівалентних векторів на наступних рівнях. Таким чином необхідно обмежити цей перебір, для запобігання дублювання.

Для цього, у випадку, коли є декілька кольорів з однаковою кількістю ребер, необхідно розглядати їх разом. В даному прикладі існує 4 фарби по 3 ребра, тобто *4*3 = 12* ребер. В такому випадку, серед ребер, які не були пофарбовані на першому рівні рекурсії вибирається *uv* ребер, які надалі необхідно буде пофарбувати у *u* кольорів.

Розглянемо процес вибору 3 ребер з 12 для розфарбування у другий колір (табл. 2).

1	1	1									
1	1		1								
1	1			1							

1	1				1						
1	1					1					
1	1						1				
1	1							1			
1	1								1		
1	1									1	
1	1										1
1		1	1								
1		1		1							
1		1			1						
1		1				1					
…											
1		1									1
1			1	1							
1			1		1						
1			1			1					
1			1				1				
…											
									1	1	1

Табл. 2

Кожен стовпець таблиці відповідає певному ребру, а кожен рядок одному з варіантів розфарбування, кількість яких рівна:

$$C_{uv}^{v} = \frac{(uv)!}{v!(uv-v)!} = \frac{uv}{v}\frac{(uv-1)!}{(v-1)!(uv-v)!} = uC_{uv-1}^{v-1} \qquad (3)$$

Як видно з формули 2 кількість варіантів завжди кратна кількості фарб. Якщо з усіх 220 варіантів розфарбування трьох з 12 ребер вибрати лише ті варіанти, в яких вибрані 3 ребра ніколи не зможуть бути розфарбовані в

інший колір, то ми забезпечимо відповідність вимозі безповторності пошуку. Для прикладу, нехай певні 3 ребра з 12 пофарбовані в перший колір. Тоді на наступному кроці 9 ребер, що залишились будуть пофарбовані у інші кольори. Таким чином жодна комбінація по 3 з цих 9 ребер не може бути вибрана як інший варіант розфарбування у перший колір, так як, в такому випадку, будуть існувати еквівалентні комбінації, в яких ці 3 ребра будуть пофарбовані у перший та інші кольори.

Забезпечити повний та безповторний перебір усіх варіантів розфарбування можна, вибравши на кожному кроці рекурсії з повного набору варіантів (таблиця 2) перші C_{uv-1}^{v-1} рядків (таблиця 3).

1	1	1									
1	1		1								
1	1			1							
…											
1	1									1	
1	1										1
1		1	1								
1		1		1							
1		1			1						
1		1				1					
…											
1		1									1
1			1	1							
1			1		1						
1			1			1					
1			1				1				
1			1					1			

1			1						1		
…											
1										1	1

Табл. 3

Такими рядками будуть ті, в яких в першому стовпчику стоїть 1. Легко побачити, що жодні 3 ребра, розфарбовані згідно таблиці 3 ніколи не будуть розфарбовані у інші кольори на наступних кроках рекурсії (що забезпечує безповторність перебору), і будь-які 3 ребра, які не були розфарбовані в перший колір точно будуть розфарбовані у інший колір (що забезпечує повноту перебору).

Визначимо кількість варіантів такого модифікованого розфарбування *uv* ребер у *u* кольорів по *v* ребра. Нагадаємо що на кожному кроці вибирається C_{uv-1}^{v-1} варіантів розфарбування. Повна кількість варіантів розфарбування:

$$C_{uv-1}^{v-1}C_{(u-1)v-1}^{v-1}C_{(u-2)v-1}^{v-1}\ldots C_{v-1}^{v-1}=\frac{C_{uv}^{v}C_{(u-1)v}^{v}C_{(u-2)v}^{v}\ldots C_{v}^{v}}{u(u-1)(u-2)\ldots 2\cdot 1}=$$

$$\frac{C_{uv}^{v}C_{(u-1)v}^{v}C_{(u-2)v}^{v}\ldots C_{v}^{v}}{u!}$$

Тобто, як було сказано вище, безповторний перебір зменшує кількість варіантів розфарбування у *u!* разів.

Таким чином, вище був описаний алгоритм, який дозволяє згенерувати усі можливі варіанти розфарбування дерева, причому ці варіанти є безповторними. Що в свою чергу дозволяє знайти точне значення верхньої межі $l_{max}(p)$.

2.3 Розфарбування за евристичним алгоритмом

В 2.2 був описаний алгоритм повного пошуку розфарбування графу, який полягав в переборі усіх можливих варіантів розфарбування графу. Одним з очевидних недоліків цього алгоритму є те, що час його виконання може бути дуже великим, що ставить задачу розробки більш швидкого евристичного алгоритму.

Далі у цій главі мною буде запропонований один з варіантів реалізації такого алгоритму, основна ідея якого полягає в послідовному розфарбуванні дерева.

Нехай є дерево, яке певним чином розфарбовано (не обов'язково оптимально), тобто кожній вершині відповідає свій колір. Візьмемо по черзі кожну вершину дерева і спробуємо перефарбувати її у інший колір. Якщо перефарбування збільшує кількість ребер, які можливо провести у графі (надалі просто кількість ребер), то залишаємо цю вершину пофарбованою в новий колір, і повторюємо алгоритм. Очевидно, що пошук припиниться, коли перефарбування жодної вершини у жодний колір не призведе до збільшення кількості ребер.

Суть евристики полягає в тому, що почергове перефарбування певної множини вершин може не призводити до збільшення кількості ребер, коли одночасне перефарбування усієї множини збільшує кількість ребер. Розглянемо приклад, зображений на рис. 13.

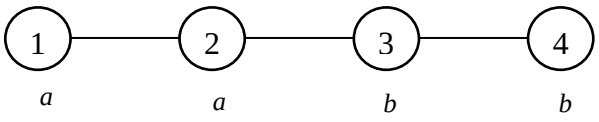


Рис. 13

Нехай є чотири вершини дерева, які розташовані на одному рівні (рис. 13), що означає, що ребра можуть бути проведені лише між сусідніми вершинами. Перші дві вершини пофарбовані в колір *a*, останні дві в колір *b*. При заданому розфарбуванні можна провести лише одне ребро: між *2-ю* і *3-*

ю вершинами (так як вони пофарбовані у різні кольори). Нескладно побачити, що змінивши колір другої вершини на *b*, а колір третьої на *a* кількість ребер стане рівною 3. Проте, якщо змінити тільки колір другої вершини, кількість ребер не зміниться (залишиться рівною 1). Аналогічно і для третьої вершини. Таким чином не можна стверджувати, що, якщо існує розфарбування з більшою кількістю ребер, то повинна існувати вершина, перефарбування якої збільшить число ребер.

Приведений вище алгоритм можна модифікувати, додавши пошук пари вершин, перефарбування яких, збільшує кількість ребер у дереві.

Твердження 2:

Якщо існує пара вершин, одночасне перефарбування яких збільшує кількість ребер у дереві, то перефарбування хоча б одної з вершин повинно не зменшувати кількість ребер.

Нехай перефарбування першої вершини зменшує число ребер на n_1, другої на n_2; n_1, $n_2 < 0$. Одночасне перефарбування вершин, може додати до кількості ребер лише одне ребро, проведене між цими двома вершинами. Тобто, кількість n_1 може збільшитись на 1, і кількість n_2 також може збільшитись на 1, проте навіть у такому випадку n_1, $n_2 \leq 0$. Тобто, в найкращому випадку, число ребер після одночасного перефарбування не зміниться, і аж ніяк не може збільшитись.

З твердження 2 випливає, що для пошуку пари вершин, потрібно розглядати тільки ті, які не зменшують число ребер.

Можна запропонувати декілька варіантів евристичного алгоритму, запропонованого в 2.3, які будуть різнитися за швидкістю пошуку, і, суб’єктивно, за якістю.

Пошук одної вершини, яка збільшує число ребер у графі, може припинятися при знаходженні першої такої вершини. Як модифікацію, можна запропонувати шукати усі вершини, і вибирати з них ту, яка максимально (або мінімально) збільшує, кількість ребер. Очевидно що другий варіант буде працювати довше.

Паралельно з пошуком одної вершини можна шукати пару вершин, яка збільшує кількість ребер, і вибирати або першу знайдену вершину (або пару), або ту вершину (або пару), яка максимально (або мінімально) збільшує кількість ребер, серед усіх знайдених вершин (або пар).

Одним із варіантів алгоритму є той, при якому пошук пари проводиться тільки у випадку, коли не була знайдена одна вершина, перефарбування якої збільшує кількість ребер. В такому випадку, при пошуку пари, слід розглядати тільки ті вершини, які при перефарбуванні не змінювали кількість ребер у графі.

Проведені автором численні тести показали, що одним з найкращих за швидкодією і якістю пошуку є наступний алгоритм: спочатку проводиться пошук усіх вершин дерева, перефарбування яких збільшує число ребер. Якщо такі знайдені – фарбуємо вершину, яка максимально збільшує кількість ребер і продовжуємо алгоритм. Якщо така вершина знайдена не була – проводиться пошук пари вершин, які збільшують кількість ребер. Вибирається перша знайдена пара. Якщо така пара не була знайдена, алгоритм завершується.

В табл. 4 наведена порівняльна характеристика результатів роботи точного і евристичного алгоритмів. Таблиця складається лише результатів роботи точного і евристичного алгоритмів, які не співпали. Були проаналізовані усі ВБС $K(m,n)$, де $4 \le n \le 22$, $3 \le m \le n - 1$, $1 \le p \le n - m + 1$. Всього комбінацій 1520, серед яких у 37 випадках результат евристичного

алгоритму не співпав з точним значенням. Усі ці випадки зазначені у таблиці.

N	M	P	R	K	Точне значення	Евристика	Δ
10	3	2	8	1	9	8	1
11	3	2	9	1	10	9	1
11	5	2	7	1	6	5	1
12	5	2	8	1	7	6	1
13	5	2	9	1	9	8	1
14	5	2	10	1	11	10	1
15	5	2	11	1	12	11	1
15	7	2	9	1	8	7	1
16	3	2	14	2	19	18	1
16	5	2	12	1	13	12	1
16	7	2	10	1	9	8	1
17	3	2	15	2	21	20	1
17	5	2	13	1	15	14	1
17	7	2	11	1	11	10	1
18	3	2	16	2	23	22	1
18	5	2	14	1	17	16	1
18	7	2	12	1	13	12	1
19	3	2	17	2	25	24	1
19	5	2	15	1	18	17	1
19	7	2	13	1	14	13	1
19	9	2	11	1	10	9	1
20	3	2	18	2	27	26	1
20	5	2	16	1	19	18	1
20	7	2	14	1	15	14	1

20	9	2	12	1	11	10	1
21	3	2	19	2	29	28	1
21	3	3	19	2	38	37	1
21	5	2	17	2	21	20	1
21	5	3	17	2	27	26	1
21	7	2	15	1	17	16	1
21	9	2	13	1	13	12	1
22	3	2	20	2	31	30	1
22	3	3	20	2	41	39	2
22	5	2	18	2	23	22	1
22	5	3	18	2	29	27	2
22	7	2	16	1	19	18	1
22	9	2	14	1	15	14	1

Табл. 4. Порівняльна таблиця евристичного і точного алгоритмів

Лише у 37 випадках з 1520 результат евристичного алгоритму не співпав з точним значенням, що свідчить про доволі високу якість евристичного алгоритму. Також, у більшості цих випадків значення евристики відрізняється всього на одиницю. Проте слід зазначити, що при збільшенні параметра N (кількості модулів ВБС) збільшується відхилення евристики від точного значення. Тому важко сказати, наскільки точним є евристичний алгоритм для більш складних ВБС з більшим числом модулів.

РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРДУКТУ

Для вирішення поставленої задачі знаходження $l_{max}(p)$ мною був розроблений програмний продукт, який включає в себе алгоритм повного пошуку, та евристичний алгоритм пошуку $l_{max}(p)$, які були детально описані в розділі 2.

Дана програма була написана на мові програмування C#.

Повний опис реалізації програми займав би багато сторінок, і навряд був би дуже інформативним, тому зупинимось лише на деяких моментах реалізації певних частин програми.

3.1 Обчислення кількості ребер

Під час пошуку $l_{max}(p)$, програма аналізує усі можливі варіанти розташування фарб у дереві, і для кожного з них підраховує кількість ребер, які можна провести при даному розфарбуванні. Для того щоб підрахувати цю кількість, необхідно перебрати усі можливі ребра, кількість яких рівна C_R^2, для кожного з яких визначити відповідність твердженню 1, і, якщо так, визначити, чи пофарбовані вершини, між якими проведене ребро, у різні кольори. Ця операція є однією з найчастіших в роботі програми, тому від швидкості її роботи напряму залежить швидкість роботи програми. Відповідність ребра твердженню 1 не залежить від розташування фарб у дереві, тому є сенс на початку роботи програми, визначити усі ребра, які відповідають твердженню 1, і лише для них виконувати перевірку розфарбування вершин у різні кольори.

3.1.1 Розпаралелювання

Зупинимось трохи детальніше на процесі пошуку $l_{max}(p)$. Після того, як користувач увів усі необхідні параметри ВБС створюється об'єкт, в якому зберігаються ці параметри і розраховуються деякі інші. Далі будується дерево ієрархії, і проводиться підготовча робота для початку пошуку. Відповідно до параметрів N, M, P створюється множина *RP*, яка складається з векторів кількості фарб *rp*. Пошук $l_{max}(p)$ відбувається відповідно до векторів *rp*, причому незалежно. Таким чином, для різних векторів *rp* пошук може відбуватися паралельно. Якщо система, в якій працює програма, має більше одного виконавчого пристрою (процесор, ядро), то є можливість пришвидшення роботи у кількість разів, рівній кількості процесорів (ядер). Ця можливість реалізована у програмі за рахунок створення декількох робочих потоків, які в подальшому системою мають розподілятися по різним виконавчим пристроям. Кожен потік вибирає з множини *RP* вектор, який ще не був початий, і виконує пошук $l_{max}(p)$ на певній підмножині усієї множини пошуку, яка відповідає вибраному вектору *rp*. Задача головного потоку синхронізувати робочі потоки і порівнювати значення $l_{max}(p)$, знайдені кожним з них. Максимальне значення і буде точним $l_{max}(p)$.

3.2 Опис інтерфейсу

Головне вікно програми зображене не рис. 14. У верхньому лівому кутку вікна знаходиться головне меню, за допомогою якого можна створити нову ВБС для пошуку $l_{max}(p)$, переключитися на режим евристичного пошуку, та зберігати результати пошуку у файл. У верхній частині вікна, під головним меню знаходиться поле параметрів ВБС та дерева ієрархії реберних функцій, до яких входять:

- N – кількість модулів ВБС.
- M – кратність відмов, до яких ВБС стійка.
- P – кількість фарб.

- R – кількість ребер мінімізованої GL-моделі (або кількість вершин дерева ієрархії реберних функцій). Цей параметр розраховується за формулою R = N – M + 1.
- K – кількість рівнів дерева ієрархії реберних функцій.

Також в цьому полі знаходиться значення розміру області пошуку, що відображає кількість усіх можливих варіантів розфарбування дерева.

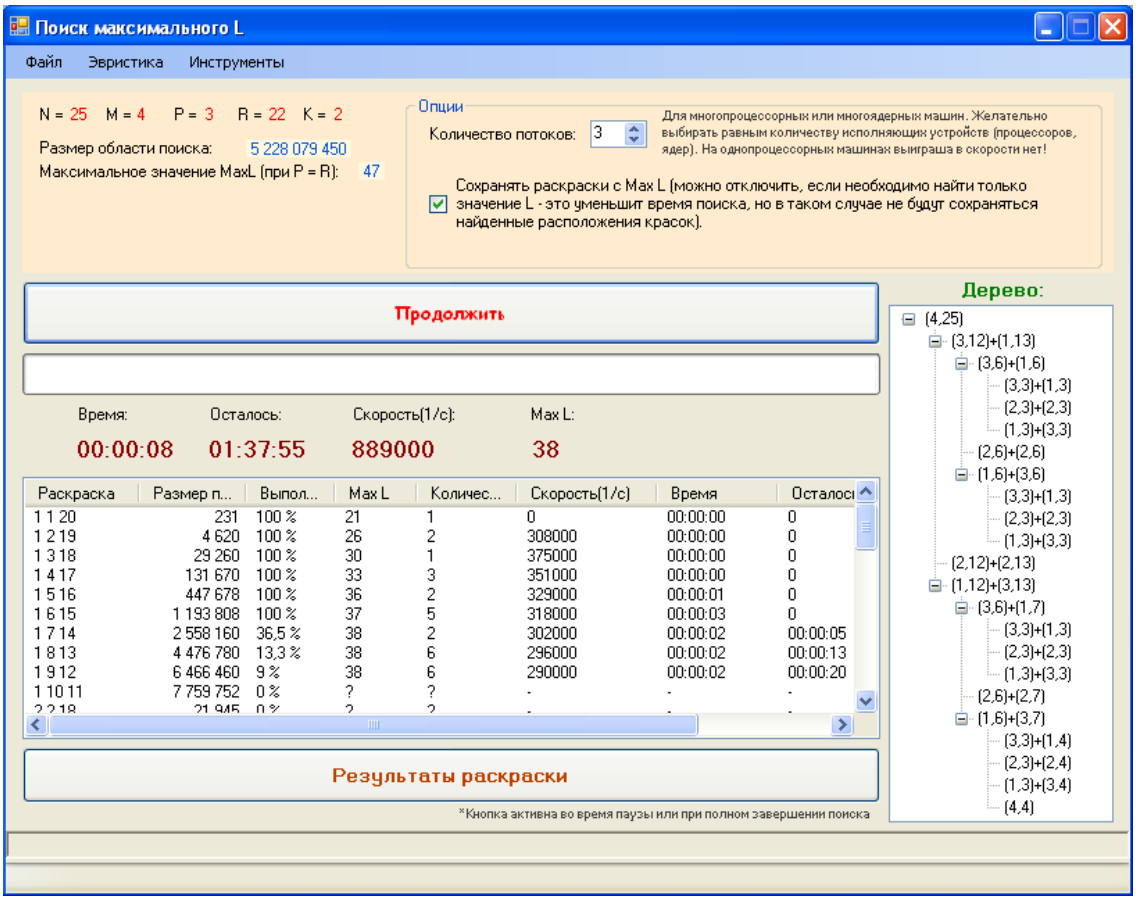


Рис 14. Головне вікно програми

В полі максимального значення *MaxL* відображається максимальне значення $l_{max}(p)$, тобто значення $l_{max}(p)$ у випадку, коли кожна вершина дерева пофарбована у свій колір. В такому випадку у дереві можливо провести усі ребра, які задовольняють твердженню 1.

Правіше від поля параметрів ВБС знаходиться поле опцій, в якому можна задати кількість робочих потоків. По замовчуванню створюється один робочий потік. Є сенс змінювати це поле у випадку роботи програми на декількох виконавчих блоках (процесорах, ядрах). Якщо у Вашому комп'ютері один одноядерний процесор – залиште це поле рівним 1, так як збільшення кількості потоків не призведе до виграшу у швидкості (навіть навпаки). Якщо в системі декілька процесорів (ядер), то це поле доцільно встановлювати рівним кількості процесорів (ядер), що призведе до пришвидшення роботи програми у кількість разів рівній кількості процесорів (ядер).

В полі опцій можна вибрати наступний параметр – чи зберігати результати розфарбування під час пошуку. По замовчуванню вона стоїть у стані «так», що призводить до збереження усіх знайдених варіантів розфарбування які дають максимальну кількість ребер. Зберігання цих варіантів забирає не дуже багато процесорного часу, проте може займати багато пам'яті (оперативної пам'яті), тому, Ви можете виключити цю опцію, якщо хочете дізнатися лише значення $l_{max}(p)$ і Вам не важливо яке саме розфарбування графу дає цей максимум.

В правій частині вікна розташоване дерево ієрархії реберних функцій.

Для створення нової ВБС потрібно в головному меню «файл» вибрати підменю «новий», після чого з'явиться діалогове вікно, в якому необхідно ввести такі параметри ВБС, як N, M, та P. Після чого програма розрахує інші параметри, побудує дерево і буде готова до початку пошуку $l_{max}(p)$. В таблиці головного вікна відображається процес пошуку для кожного вектора *rp*. Перший стовпчик відображає вектор *rp*, в другому знаходиться кількість варіантів розфарбування дерева для конкретного *rp,* третій стовпчик показує скільки відсотків розфарбувань від загальної кількості розглянуто, в четвертому стовпчику знаходиться значення *MaxL* – максимальна кількість

ребер для даного вектору *rp,* в п’ятому кількість варіантів розташування фарб у дереві, які дають *MaxL,* в останніх стовпчиках відображаються поточна швидкість пошуку, пройдений час, та час, який залишився до закінчення пошуку *MaxL* для конкретного вектору *rp.*

Для початку пошуку $l_{max}(p)$ необхідно натиснути кнопку «старт», після чого програма почне працювати. Під кнопкою знаходиться рядок поточного стану пошуку, в якому відображається скільки відсотків варіантів розфарбування графу вже оброблено.

Трохи нижче розташовані такі параметри пошуку, як час пошуку, поточна швидкість пошуку, час який залишився, та поточне значення $l_{max}(p)$. В кінці роботи програми, саме в цьому полі буде знаходитись точне значення $l_{max}(p)$.

В будь-який момент часу можна призупинити роботу програми, змінити деякі параметри, переглянути поточні варіанти розфарбування та продовжити роботу.

Коли програма завершила пошук, результати розфарбування можна побачити в іншому вікні, яке відкривається натисненням на кнопку «результати» (рис. 15). В ньому можна переглянути усі знайдені варіанти розташування фарб, при яких в дереві можна провести максимальну кількість ребер $l_{max}(p)$ (лише у випадку коли в опціях був вибрано зберігати результати розфарбування). У лівому списку знаходяться вектора *rp*, в правому власне вектора розташування фарб, при виборі яких, дерево, розташоване в правій частині вікно розфарбовується згідно вибраного вектору.

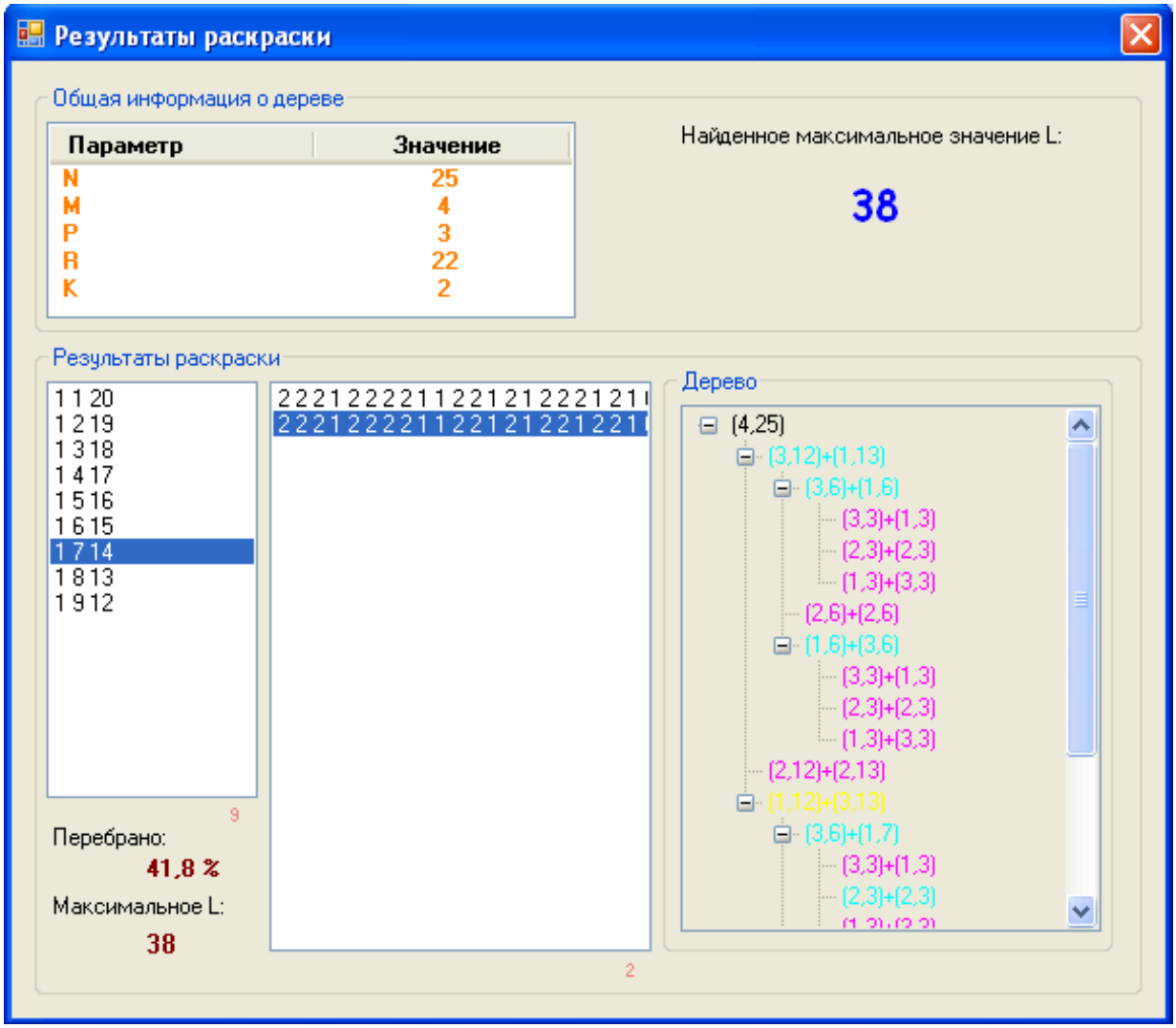


Рис 15. Вікно результатів програми

Для пошуку розфарбування евристичним алгоритмом, в головному меню головного вікна необхідно вибрати пункт «евристика», після чого з’явиться вікно, зображене на рис. 16.

Далі, для створення нової ВБС для евристичного пошуку необхідно вибрати пункт «файл» головного меню вікна евристичного пошуку і ввести відповідні параметри N, M та P. Після чого натиснути кнопку старт. В полі нижче кнопки буде показаний єдиний поточний варіант розфарбування

графу, вибір якого призведе до розфарбування дерева, розташованого в правій частині вікна.

Результати роботи алгоритму повного пошуку і евристичного алгоритму можна зберігати у текстовий файл – для цього необхідно в головному меню відповідного вікна вибрати підменю «інструменти» і вибрати пункт «зберегти у файл». Після з’явиться діалогове вікно, в якому необхідно буде ввести ім’я файлу.

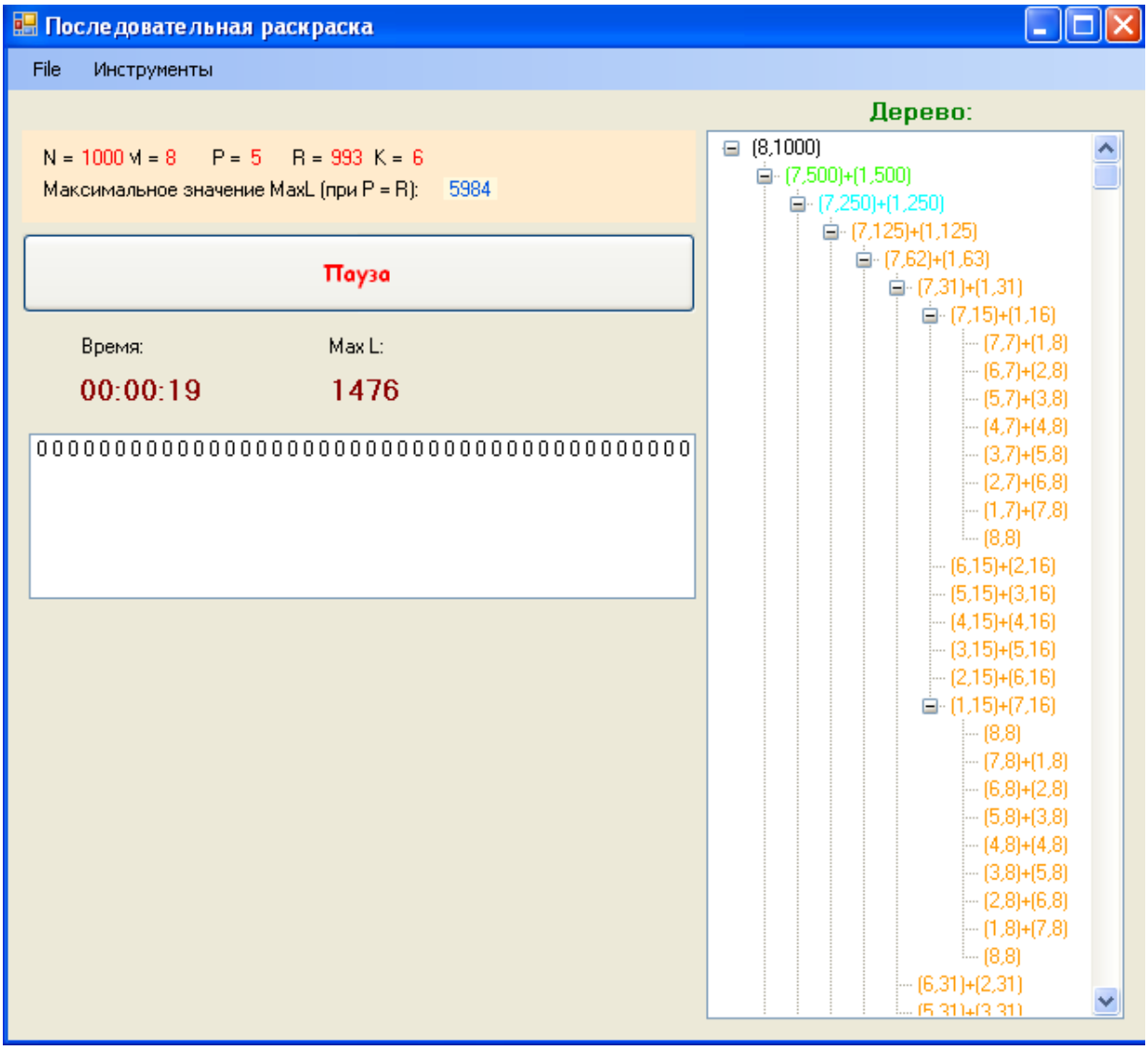


Рис 16. Вікно евристичного пошуку

ВИСНОВКИ

Розроблене програмне забезпечення дозволяє знайти верхню межу $l_{max}(p)$, на основі якої можна оцінити границі, в яких знаходиться кількість додаткових ребер, які необхідно провести для перетворення базової GL-моделі. Показано, яким чином цю задачу можна звести до задачі розфарбування графу.

Були розроблені та реалізовані два алгоритми пошуку $l_{max}(p)$: алгоритм повного пошуку, який дає точне значення, та евристичний алгоритм, перевагою якого є більш висока швидкість роботи програми, при доволі високій якості пошуку.

Програмний продукт реалізовано таким чином, що в процесі пошуку $l_{max}(p)$ створюються незалежні частини програми, які можуть паралельно працювати на різних виконавчих пристроях, що дає змогу виконувати програму практично на будь-якій кількості процесорів, чим багатократно підвищувати швидкість її роботи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Романкевич В.О., Потапова К.Р., Бахтари Хедаятоллах, Назаренко В.В. GL-модель поведінки відмовостійких багатопроцесорних систем з мінімальним числом ребер, що втрачаються // Вісник НТУУ “КПІ” Інформатика, управління та обчислювальна техніка. - 2006. №45. - с.93 - 100.
2. Романкевич О.М., Іванов В.В., Романкевич В.О. Аналіз відмовостійких багатомодульних систем зі складним розподілом відмов на основі циклічних GL-моделей // Електронне моделювання. - №5, т.26, 2004, с.67-81.
3. Романкевич О.М., Карачун Л.Ф., Романкевич В.О. Графологічні моделі для аналізу складних відмовостійких обчислювальних систем // Електронне моделювання. - 2001.-т.23, №1.- С.102-111.
4. Романкевич В.О. Про один спосіб побудови GL-моделей відмовостійких багатопроцесорних систем // Радіоелектронні і комп‘ютерні системи. - №7, 2006, с.47-51

ДОДАТКИ. КОПІЇ ГРАФІЧНИХ МАТЕРІАЛІВ

ІАЛЦ.466500.005 Д1	Побудова дерева ієрархії реберних функцій. Схема алгоритму.
ІАЛЦ.466500.006 Д2	Повний пошук розфарбування графу. Схема алгоритму.
ІАЛЦ.466500.007 Д3	Евристичний пошук розфарбування графу. Схема алгоритму.
ІАЛЦ.466500.008 Д4	Програма розфарбування графу. Схема структурна.